Algorithmique et Structures de Données 1
Projet 2 de Travaux Pratiques

Année 2018/2019

- Introduction

L'objet de ce projet est d'étudier deux ou trois algorithmes de tri utilisables sur des chaînages. Chaque algorithme devra être implémenté, vérifié sur des exemples soigneusement choisis et comparé aux autres.

L'ordre choisi ici est un ordre lexicographique : d'abord en regardant le caractère, puis en cas d'égalité de caractère, en fonction de l'entier, et si le caractère et l'entier sont égaux, selon le réel.

Le travail est à effectuer en binôme (deux étudiant(e)s) et le code doit être compilable et exécutable sur les machines des salles de TP — vous pouvez travailler sur une machine personnelle mais **devez** vérifier le fonctionnement en salle de TP..

A - Structure

La structure de chaînage proposée est la suivante, reprise du TP précédent :

```
struct t_voie {
       char   typeVoie ;
       int numero;
       double trafic;
       t_voie *autre; // lien vers une autre voie
};

typedef t_voie * ptr_voie ;
```

`t_voie` représente ainsi un maillon de chaînage simple, son champs `autre` valant `nullptr` pour le dernier maillon.

Un chaînage vide est donc représenté par la valeur `nullptr`. Les algorithmes de tri demandés nécessitent quelques fonctions auxiliaires à écrire :

1. Écrire une fonction **`plusPetitOuEgal`** prenant en paramètres deux `ptr_voie` et précisant si le premier est situé avant le second dans l'ordre lexicographique choisi, ou lui est égal. Bien préciser les préconditions.
2. Écrire une procédure **`mettreAvant`** prenant en paramètre deux `ptr_voie` et plaçant le maillon pointé par le premier en tête du chaînage commençant par le second. Bien préciser les préconditions.
3. Écrire une fonction **`retraitPremier`** prenant en paramètre modifiable un `ptr_voie` et enlevant le premier maillon du chaînage commençant par ce pointeur ; cette fonction renvoie le pointeur vers le maillon retiré. Bien préciser les préconditions.

B - Test

Avant de continuer, il est impératif de vérifier que le code déjà produit est correct :

4. Écrire un programme principal permettant de tester les procédures et fonctions écrites. Il peut être intéressant de réutiliser le travail du TP précédent (création et affichage d'un chaînage en particulier). Bien présenter les jeux d'essai.

C - Tri insertion

Le principe du tri par insertion est de prendre chaque maillon du chaînage à trier et de l'insérer à sa place dans un nouveau chaînage maintenu trié.

5. Écrire une procédure **`placerEnOrdre`** prenant en paramètres deux `ptr_voie` et insérant le maillon désigné par le premier à sa place dans le chaînage commençant par le second. Avant exécution, le chaînage est supposé trié par ordre croissant (chaque maillon est **`plusPetitOuEgal`** que le suivant), après exécution il doit encore l'être (post-condition).
6. Écrire une seconde version de **`placerEnOrdre`** de façon à disposer d'une version itérative et d'une version récursive.
7. Écrire ensuite une procédure **`trierParInsertion`** prenant en paramètre modifiable un `ptr_voie` désignant un chaînage de voies et triant ce chaînage. Pour cela, créer un chaînage vide et insérer dedans (en ordre) successivement tous les maillons du chaînage de départ. Mettre à jour à la fin le pointeur passé en paramètre.
8. Bien tester toutes ces fonctions/procédures. Discuter sur le rapport des cas particuliers. Ce tri est-il stable ? Détailler le coût théorique de la version itérative en supposant que le chaînage fourni au départ est trié en ordre inverse.

D - Tri fusion

Le principe du tri fusion est de séparer le chaînage en deux chaînages de tailles proches, de trier récursivement chacun, puis de les rassembler intelligemment (fusion).

9. Écrire une fonction **`unSurDeux`** prenant en paramètre un `ptr_voie` donnant le début d'un chaînage et retournant un pointeur sur un nouveau chaînage contenant la moitié du chaînage donné (le second maillon, le quatrième, le sixième, et ainsi de suite). Les maillons en questions doivent être enlevés du chaînage fourni. Justifier la signature de la fonction et bien définir la précondition (discuter des cas pathologiques).
10. Écrire la fonction **`fusion`** prenant en paramètres deux `ptr_voie` désignant deux chaînages supposés triés et retournant un `ptr_voie` commençant un chaînage trié contenant tous les maillons des deux chaînages. Pour cela, prendre (et enlever) le premier maillon le plus petit entre les deux chaînages, l'insérer en queue du chaînage résultat en construction, et recommencer jusqu'à épuisement de l'un des deux. Les deux paramètres doivent valoir `nullptr` après exécution.
11. Écrire une seconde version de **`fusion`** de façon à disposer d'une version itérative et d'une version récursive.
12. En déduire une procédure **`trierParFusion`** prenant en paramètre modifiable un `ptr_voie` désignant un chaînage de voies et triant ce chaînage. Pour cela, couper le chaînage en deux, trier récursivement chaque moitié puis fusionner les deux résultats. Mettre à jour à la fin le pointeur passé en paramètre.
13. Bien tester toutes ces fonctions/procédures. Discuter sur le rapport des cas particuliers. Ce tri est-il stable ? Calculer le coût théorique en supposant que le nombre de maillons du chaînage original est une puissance de 2 (tout découpage donne des chaînages de même longueur) et que la fusion nécessite de prendre alternativement dans chacun des deux chaînages. Le calcul donne une formule de récurrence Coût(n) en fonction de Coût(n/2) qu'il faut ensuite évaluer dans le cas général (éliminer la récurrence).

E - Comparaison

Une fois les questions précédentes traitées,

14. Écrire une fonction **`voieAleatoire`** sans paramètre et fournissant un pointeur sur un `t_voie` nouvellement alloué et valué aléatoirement. Pour cela remarquer que `(char)('A'+rand()%26)` fournit une lettre au hasard.
15. Écrire une fonction **`chainageAleatoire`** prenant un entier n en paramètre et fournissant un chaînage de n maillons valués aléatoirement.
16. En utilisant les fonctions précédentes, étudier expérimentalement l'évolution du temps de calcul en fonction du nombre de maillons des deux procédures de tri créés. Pour chacune, comparer au coût théorique ; puis les comparer entre elles.
17. Éventuellement (bonus) proposer (et si possible implémenter) une troisième méthode de tri manipulant des chaînages (sans tableau donc :-). La comparer aux deux autres.

F - Travail à rendre

Le travail d'implémentation sera réalisé dans un unique fichier source, clairement présenté et documenté, comportant l'ensemble des procédures et fonctions réalisées ainsi qu'un programme principal démontrant leur utilisation.

Vous rédigerez un document de travail au préalable et en parallèle, précisant les ambiguïtés du sujet, les difficultés rencontrées. Il servira de base à votre rapport.

Ce document sera complété au fur et à mesure avec les choix effectués (constantes, comportement des procédures et fonctions non précisé dans l'énoncé, …) et les limites d'utilisation.

Le rapport final comportera aussi les réponses aux questions de l'énoncé (8 et 13) et les graphiques d'études des tris.

Il devra être bien rédigé (français correct, attention à l'orthographe !) et agréable à lire, comporter introduction et conclusion et éventuellement des annexes pour les détails techniques (programmes, tests réalisés, tableaux de valeurs, ...).

Le rapport (au format PDF) et le code C++ (fichier source .cpp) sont à déposer
sur MADOC dans un répertoire compacté (zip ou tar.gz)

au plus tard le **vendredi 21 décembre 2018 à 19h00**.

Un seul rendu par binôme (sur l'un ou l'autre des deux comptes).